UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/688,573	10/20/2003	Robert M. Zeidman	ZEID-01	2483

66323 7590 09/11/2008
ZEIDMAN TECHNOLOGIES, INC.
15565 SWISS CREEK LANE
CUPERTINO, CA 95014

EXAMINER	
WANG, BEN C	

ART UNIT	PAPER NUMBER
2192	

MAIL DATE	DELIVERY MODE
09/11/2008	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

Office Action Summary	Application No.	Applicant(s)	
	10/688,573	ZEIDMAN, ROBERT M.	
	Examiner	**Art Unit**	
	BEN C. WANG	2192	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *27 May 2008*.

2a)☐ This action is **FINAL**. 2b)☒ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1,3-7,15,17-22 and 24-28* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1, 3-7, 15, 17-22, and 24-28* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☐ The drawing(s) filed on _____ is/are: a)☐ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)

2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)

3) ☐ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date _____.

4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____ .

5) ☐ Notice of Informal Patent Application

6) ☐ Other: _____.

DETAILED ACTION

1. Applicant's amendment dated May 27, 2008, responding to the March 13, 2008 Office action provided in the rejection of claims 1, 3-7, 15, 17-22, and 24-28.

Claims 1, 3-7, 15, 17-22, and 24-28 remain pending in the application and which have been fully considered by the examiner.

Applicant's arguments with respect to claims rejection have been fully considered but are moot in view of the new grounds of rejection – see *Liu et al.*, art made of record, as applied hereto.

Claim Rejections – 35 USC § 103(a)

The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made

2. Claims 1, 3, 5, 7, 15, 17, 19, 21-22, 24, 26, and 28 are rejected under 35 U.S.C. 103(a) as being unpatentable over Lehman et al. (US Patent 4,796,179) (hereinafter 'Lehman') in view of Gauthier et al. *'Automatic Generation and Targeting of Application Specific Operating Systems and Embedded Systems Software'*, 2001, IEEE (hereinafter 'Gauthier') and Liu et al. *'Timed Multitasking for Real-Time Embedded Software'*, February 2003, IEEE (hereinafter 'Liu' – art made of record)

3. **As to claim 1** (Previously Presented), Lehman discloses a method for developing a real-time operating system (e.g., Fig. 1; Col. 1, lines 46-48; Col. 4, lines 63-68; Col. 5, lines 1-2), comprising: specifying a set of *n* tasks (e.g., Col. 1, lines 33-38), *task(1)* through *task(n),* to be scheduled for execution; (e.g., Abstract, lines 8-14; Col. 3, lines 1-8; Col. 5, lines 5-12; Col. 135, lines 17-24).

Lehman also discloses specifying a scheduling algorithm (e.g., Col.3, lines 36-39; Col. 9, lines 56-61; Col. 32, lines 44-47; Col. 20, line 63 through Col. 21, line 20; Col.7, lines 29-32; Col. 16, lines 21-23; Col. 2, lines 36-39; Col. 9, lines 62-68; Col. 10, lines 1-2; Col. 32, lines 5-54) for scheduling the execution of the set of *n* tasks.

Although Lehman discloses at least one of the task of the set of *n* tasks being selected as a preemptive or a non-preemptive task (e.g., Col. 9, lines 52-56; Col. 10, Lines 3-12; Col. 35, Lines 1-5; Col. 136, Lines 40-50); synthesizing source code to implement a task scheduler (e.g., Fig. 4, element 24; Fig. 24; Fig. 26; Col. 2, lines 36-39; Col. 3, lines 36-39; Col. 9, lines 56-61; Abstract, lines 20-25; Col. 10, lines 8-12) that uses the scheduling algorithm for controlling execution of said *n* tasks, Lehman does not explicitly disclose synthesizing source code from commands embedded in source code to implement the task scheduler that uses said scheduling algorithm for controlling execution of said set of n tasks, said synthesized source code being executable on a target system after compilation.

However, in an analogous art of '*Automatic Generation and Targeting of Application Specific Operating Systems and Embedded Systems Software*', Gauthier

discloses synthesizing source code (e.g., Sec. 4.2 – Synthesis of Application Specific OS and SW targeting) from commands embedded in source code (e.g., Fig. 6 – an example of macro code expansion; Sec. 3.5.3 – Code Expander, 1st Para – Code Expander takes as input a list of macro code from Code Selector and parameters (processor and allocation information) from Architecture Analyzer; it generates the final OS code by expanding the macro codes of elements to source codes (in C or assembly); 2nd Para, Lines 19-22, Figure 6(b) shows an example of expanded code in C for this case; note that, in this case, another scheduler can be selected to schedule tasks that have different priority values) to implement the task scheduler that uses said scheduling algorithm for controlling execution of said set of n tasks (e.g., Fig. 1 – an example of OS-based SW implementation of multiple tasks – “scheduling”; Sec. 2 – Related Work, 2nd Para – there are three approaches in SW implementation from multi-task descriptions; the first two approaches user OS as a scheduler and an interface of multiple tasks to the target architecture; bullets 1 through 3), said synthesized source code being executable on a target system after compilation (e.g., Fig. 3 – a flow of automatic generation of application specific OS and automatic SW targeting, elements of “Operating System Library”, “ Code Selector”, “Code Expander”, “Targeted Operating Systems Code”; Sec. 3.7 – Makefile Generator – Makefile Generator takes as input (1) processor type information form Architecture Analyzer, (2) a list of source codes of OS (in C and assembly) from elements of Code Selector and (3) a list of the application SW codes; it determines the right compiler and linker and generates a makefile (for each processor) that includes the two code lists of OS and application SW).

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Gauthier into the Lehman's system to further provide synthesizing source code from commands embedded in source code to implement the task scheduler that uses said scheduling algorithm for controlling execution of said set of n tasks, said synthesized source code being executable on a target system after compilation in Lehman system.

The motivation is that it would enhance the Lehman's system by taking, advancing and/or incorporating Gauthier's system which offers significant advantages for a method of automatic generation of application specific operating systems (OS's) and automatic targeting of application software as once suggested by Gauthier (e.g., Abstract, Lines 1-3).

Furthermore, Lehman and Gauthier do not explicitly disclose specifying t init-tasks that are executed only once upon initial execution of a task scheduler, t being less than or equal to n; the task scheduler further controlling one execution of each of said set of t init-tasks.

However, in an analogous art of *Timed Multitasking for Real-Time Embedded Software*, Liu discloses specifying t init-tasks that are executed only once upon initial execution of a task scheduler, t being less than or equal to n (e.g., P. 71, 'Code Generation' – 3rd Para - … An ISR is synthesized as <u>an independent thread</u>, w<u>ith an init() method that initializes hardware resources</u> and a start() method that registered itself to the run-time system …; 4th Para - <u>Tasks</u> … <u>to init</u>() and start() … stopExec() … A <u>TASK data structure</u> is also generated, <u>which defines a scheduling entry representing</u>

a task …; Fig. 10 – Definition of a TASK in generated C code); the task scheduler further controlling one execution of each of said set of t init-tasks (e.g., P. 71, 'Scheduling Analysis' - … be fed to scheduling algorithms for schedulability analysis and priority assignment; 'Code Generation', 1st Para - … providing the interface and scheduling code, the actor code becomes single threaded and self-contained …)

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Liu into the Lehman-Gauthier's system to further provide specifying t init-tasks that are executed only once upon initial execution of a task scheduler, t being less than or equal to n the task scheduler further controlling one execution of each of said set of t init-tasks in the Lehman-Gauthier system.

The motivation is that it would further enhance the Lehman-Gauthier's system by taking, advancing and/or incorporating Liu's system which offers significant advantages that this model takes advantage of an actor-oriented software architecture and embraces timing properties at design time, so that designers can specify when the computational results are produced to the physical world or to other actors as once suggested by Liu (e.g., P. 66, 1st Para)

4. **As to claim 3** (Previously Presented), Lehman discloses the method and the apparatus further including specifying *f f-loop* tasks, each having an associated integer value *li* for *i* ranging from *1* to *f* and *f* being less than or equal to *n* (e.g., Col. 20, line 63 through Col. 21, line 20 – for loops using an incrementing or decrementing counter, i.e.

Loop for *I* = 1 to *X* (executing) block of statements), said task scheduler addresses the task scheduler executing the loops including a continuously executing loop such that each *f-loop* task executes exactly once every *li* times that the loop is executed (e.g., Col. 21, lines 13-19)

5. **As to claim 5** (Previously Presented), Lehman discloses the method and apparatus further including specifying *c call-tasks*, *c* being less than or equal to *n*, the task scheduler scheduling a *call-task* when another task requests that the *call-task* be executed (e.g., Col. 7, lines 29-32; Col. 16, lines 21-23).

6. **As to claim 7** (Previously Presented), Lehman discloses the method and the apparatus where tasks are given priority values such that whenever the task scheduler chooses between scheduling multiple tasks, all of which being ready to be executed, said task scheduler chooses from among those tasks that have the highest priority values (e.g., Col. 2, lines 36-39; Col. 9, lines 62-68; Col. 10, lines 1-2; Col. 32, lines 5-54)

7. **As to claim 15** (Previously Presented), Lehman discloses an apparatus for developing a real-time operating system comprising: a computer (e.g., Fig. 27 – multi-process controller, Lines 30-32); a computer readable medium in data communication with the computer (e.g., Col. 135, Line 15 through Col. 137, Lines 64), the computer readable medium including a software synthesis program stored thereon (e.g., Col. 135,

Line 15 through Col. 137, Lines 64), which when executed by the computer causes the computer

Although Lehman discloses to specify a set of *n* tasks (e.g., Col. 1, lines 33-38), task (1) through task (n), to be scheduled for execution; specify a scheduling algorithm (e.g., Col.3, lines 36-39; Col. 9, lines 56-61; Col. 32, lines 44-47; Col. 20, line 63 through Col. 21, line 20; Col.7, lines 29-32; Col. 16, lines 21-23; Col. 2, lines 36-39; Col. 9, lines 62-68; Col. 10, lines 1-2; Col. 32, lines 5-54) for scheduling the execution of the set of *n* tasks; and synthesize source code with to implement a task scheduler (e.g., Fig. 4, element 24; Fig. 24; Fig. 26; Col. 2, lines 36-39; Col. 3, lines 36-39; Col. 9, lines 56-61; Abstract, lines 20-25; Col. 10, lines 8-12) that uses the scheduling algorithm and for controlling execution of the set of *n* tasks; but does not explicitly disclose synthesizing source code from commands embedded in source code to implement the task scheduler, and synthesized source code being executable on a target system after compilation.

However, in an analogous art of '*Automatic Generation and Targeting of Application Specific Operating Systems and Embedded Systems Software*', Gauthier discloses synthesizing source code from commands embedded in source code to implement the task scheduler (e.g., Fig. 6 – an example of macro code expansion; Sec. 3.5.3 – Code Expander, 1st Para – Code Expander takes as input a list of macro code from Code Selector and parameters (processor and allocation information) from Architecture Analyzer; it generates the final OS code by expanding the macro codes of elements to source codes (in C or assembly); 2nd Para, Lines 19-22, Figure 6(b) shows

an example of expanded code in C for this case; note that, in this case, another scheduler can be selected to schedule tasks that have different priority values), and synthesized source code being executable on a target system after compilation (e.g., Fig. 3 – a flow of automatic generation of application specific OS and automatic SW targeting, elements of “Operating System Library”, “ Code Selector”, “Code Expander”, “Targeted Operating Systems Code”; Sec. 3.7 – Makefile Generator – Makefile Generator takes as input (1) processor type information form Architecture Analyzer, (2) a list of source codes of OS (in C and assembly) from elements of Code Selector and (3) a list of the application SW codes; it determines the right compiler and linker and generates a makefile (for each processor) that includes the two code lists of OS and application SW)

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Gauthier into the Lehman’s system to further provide synthesizing source code from commands embedded in source code to implement the task scheduler, and synthesized source code being executable on a target system after compilation in Lehman system.

The motivation is that it would enhance the Lehman’s system by taking, advancing and/or incorporating Gauthier’s system which offers significant advantages for a method of automatic generation of application specific operating systems (OS’s) and automatic targeting of application software as once suggested by Gauthier (e.g., Abstract, Lines 1-3)

Furthermore, Lehman and Gauthier do not explicitly disclose specifying t init-tasks that are executed only once upon initial execution of a task scheduler, t being less than or equal to n; the task scheduler further controlling one execution of each of said set of t init-tasks.

However, in an analogous art of *Timed Multitasking for Real-Time Embedded Software*, Liu discloses specifying t init-tasks that are executed only once upon initial execution of a task scheduler, t being less than or equal to n (e.g., P. 71, 'Code Generation' – 3rd Para - … An ISR is synthesized as an independent thread, with an init() method that initializes hardware resources and a start() method that registered itself to the run-time system …; 4th Para - Tasks … to init() and start() … stopExec() … A TASK data structure is also generated, which defines a scheduling entry representing a task …; Fig. 10 – Definition of a TASK in generated C code); the task scheduler further controlling one execution of each of said set of t init-tasks (e.g., 'Scheduling Analysis' - … be fed to scheduling algorithms for schedulability analysis and priority assignment; 'Code Generation', 1st Para - … providing the interface and scheduling code, the actor code becomes single threaded and self-contained …)

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Liu into the Lehman-Gauthier's system to further provide specifying t init-tasks that are executed only once upon initial execution of a task scheduler, t being less than or equal to n; the task scheduler further controlling one execution of each of said set of t init-tasks in the Lehman-Gauthier system.

The motivation is that it would further enhance the Lehman-Gauthier's system by taking, advancing and/or incorporating Liu's system which offers significant advantages that this model takes advantage of an actor-oriented software architecture and embraces timing properties at design time, so that designers can specify when the computational results are produced to the physical world or to other actors as once suggested by Liu (e.g., P. 66, 1st Para)

8. **As to claim 17** (Previously Presented), Lehman discloses the apparatus being further configured to specify *f f-loop* tasks, each having an associated integer value *I(i)* for *i* ranging from *1* to *f* and *f* being less than or equal to *n* (e.g., Col. 20, line 63 through Col. 21, line 20 – for loops using an incrementing or decrementing counter, i.e. Loop for *I* = 1 to *X* (executing) block of statements), the task scheduler including a continuously executing loop such that each *f-loop* task executes exactly once every *I(i)* times that the loop is executed (e.g., Col. 21, lines 13-19)

9. **As to claim 19** (Previously Presented), please refer to above claim **5** accordingly.

10. **As to claim 21** (Previously Presented), please refer to above claim **7** accordingly.

11. **As to claim 22** (Previously Presented), Lehman discloses an apparatus for developing a real-time operating system (e.g., Fig. 1; Col. 1, lines 46-48; Col. 4, lines 63-68; Col. 5, lines 1-2) comprising: means for specifying a set of n tasks (e.g., Col. 1, lines 33-38), task (1) through task (n), to be scheduled for execution (e.g., Abstract, lines 8-14; Col. 3, lines 1-8; Col. 5, lines 5-12; Col. 135, lines 17-24); means for specifying a set of n tasks, means for specifying a scheduling algorithm for scheduling the execution of said the of *n* tasks (e.g., Col.3, lines 36-39; Col. 9, lines 56-61; Col. 32, lines 44-47; Col. 20, line 63 through Col. 21, line 20; Col.7, lines 29-32; Col. 16, lines 21-23; Col. 2, lines 36-39; Col. 9, lines 62-68; Col. 10, lines 1-2; Col. 32, lines 5-54)

Although Lehman discloses a) means for specifying a set of n tasks, task (1) through task (n), to be scheduled for execution, at least one of the tasks of the set of n tasks being a preemptive or a non-preemptive task; c) means for synthesizing source code with to implement the task scheduler that uses said scheduling algorithm and said for controlling execution of the set of *n* tasks; Lehman does not explicitly disclose means for synthesizing source code from commands embedded in source code to implement a task scheduler that uses said scheduling algorithm for controlling execution of said set of n tasks, said synthesized source code being executable on a target system after compilation.

However, in an analogous art of '*Automatic Generation and Targeting of Application Specific Operating Systems and Embedded Systems Software*', Gauthier discloses means for synthesizing source code from commands embedded in source code to implement the task scheduler that uses said scheduling algorithm for controlling

execution of said set of n tasks (e.g., Fig. 6 – an example of macro code expansion; Sec. 3.5.3 – Code Expander, 1st Para – Code Expander takes as input a list of macro code from Code Selector and parameters (processor and allocation information) from Architecture Analyzer; it generates the final OS code by expanding the macro codes of elements to source codes (in C or assembly); 2nd Para, Lines 19-22, Figure 6(b) shows an example of expanded code in C for this case; note that, in this case, another scheduler can be selected to schedule tasks that have different priority values; Fig. 1 – an example of OS-based SW implementation of multiple tasks – "scheduling"; Sec. 2 – Related Work, 2nd Para – there are three approaches in SW implementation from multi-task descriptions; the first two approaches user OS as a scheduler and an interface of multiple tasks to the target architecture; bullets 1 through 3), said synthesized source code being executable on a target system after compilation (e.g., Fig. 3 – a flow of automatic generation of application specific OS and automatic SW targeting, elements of "Operating System Library", " Code Selector", "Code Expander", "Targeted Operating Systems Code"; Sec. 3.7 – Makefile Generator – Makefile Generator takes as input (1) processor type information form Architecture Analyzer, (2) a list of source codes of OS (in C and assembly) from elements of Code Selector and (3) a list of the application SW codes; it determines the right compiler and linker and generates a makefile (for each processor) that includes the two code lists of OS and application SW)

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Gauthier into Lehman's system to further provide means for synthesizing source code from commands

embedded in source code to implement the task scheduler that uses said scheduling algorithm for controlling execution of said set of n tasks, said synthesized source code being executable on a target system after compilation in the Lehman system.

The motivation is that it would enhance the Lehman's system by taking, advancing and/or incorporating Gauthier's system which offers significant advantages for a method of automatic generation of application specific operating systems (OS's) and automatic targeting of application software as once suggested by Gauthier (e.g., Abstract, Lines 1-3)

Furthermore, Lehman and Gauthier do not explicitly disclose means for specifying t init-tasks that are executed only once upon initial execution of a task scheduler, t being less than or equal to n; the task scheduler further controlling one execution of each of said set of t init-tasks.

However, in an analogous art of *Timed Multitasking for Real-Time Embedded Software*, Liu discloses specifying t init-tasks that are executed only once upon initial execution of a task scheduler, t being less than or equal to n (e.g., P. 71, 'Code Generation' – 3rd Para - … An ISR is synthesized as <u>an independent thread</u>, w<u>ith an init() method that initializes hardware resources</u> and a start() method that registered itself to the run-time system …; 4th Para - <u>Tasks</u> … <u>to init</u>() and start() … stopExec() … A <u>TASK data structure</u> is also generated, <u>which defines a scheduling entry representing a task</u> …; Fig. 10 – Definition of a TASK in generated C code); the task scheduler further controlling one execution of each of said set of t init-tasks (e.g., P. 71, 'Scheduling Analysis' - … be fed to scheduling algorithms for <u>schedulability analysis and</u>

priority assignment; ‘Code Generation’, 1st Para - … providing the interface and scheduling code, the actor code becomes single threaded and self-contained …)

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Liu into the Lehman-Gauthier’s system to further provide means for specifying t init-tasks that are executed only once upon initial execution of a task scheduler, t being less than or equal to n; the task scheduler further controlling one execution of each of said set of t init-tasks in the Lehman-Gauthier system.

The motivation is that it would further enhance the Lehman-Gauthier’s system by taking, advancing and/or incorporating Liu’s system which offers significant advantages that this model takes advantage of an actor-oriented software architecture and embraces timing properties at design time, so that designers can specify when the computational results are produced to the physical world or to other actors as once suggested by Liu (e.g., P. 66, 1st Para)

12. **As to claim 24** (Previously Presented), please refer to above claim **3** accordingly.

13. **As to claim 26** (Previously Presented), please refer to above claim **5** accordingly.

14. **As to claim 28** (Previously Presented), please refer to above claim **7** accordingly.

15. Claims 4, 18, and 25 are rejected under 35 U.S.C. 103(a) as being unpatentable over Lehman, in view of Gauthier, Liu and Xu et al. *On Satisfying Timing Constraints in Hard-Real-Time Systems'*, 1991, ACM' (hereinafter 'Xu')

16. **As to claim 4** (Previously Presented), Lehman discloses the method and apparatus including means for specifying "loops" mechanism (e.g., Col. 20, line 63 through Col. 21 line 20)

But, Lehman, Gauthier and Liu do not specifically disclose *p-loop* task.

However, in an analogous art, Xu discloses means for specifying *p-loop* tasks, each having an associated integer value *ti* for *i* ranging from *1* to *p* and *p* being less than or equal to *n*, the number *ti* representing a number of regular time units (e.g., Sec. 2, 3rd paragraph, lines 1-4), said task scheduler including a timer that schedules each *p-loop* task i to be executed approximately once every *ti* time units (e.g., Sec. 2, 3rd paragraph, lines 1-4; Sec. 2, 7th paragraph, on page 133 – A periodic process *p* can be described by a quadruple(*rp, cp, dp, prdp*), where *prdp* is the period, *cp* is the worse case computation time required by process *p*, *dp* is the deadline, *rp* is the release time).

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to combine the teachings of Lehman, Gauthier and Liu, and the teachings of Xu in order to provide a timing constraints mechanism in Lehman-Gauthier-Liu system.

The motivation is that (a) pre-run-time scheduling is essential if we want to guarantee that timing constraints will be satisfied in a complex hard-real-time system, (b) appropriate algorithms for solving mathematical scheduling problems that address those concerns can be used to automate pre-run-time scheduling, (c) if the task of computing schedules is completely automated, it would be very easy to modify the system and re-compute new schedules in case changes are required by applications as once suggested by Xu (e.g., Abstract, Lines 1-3; Sec. 6, 3rd Para., 6th Para.)

17. **As to claim 18** (Previously Presented), please refer to above claim **4** accordingly.

18. **As to claim 25** (Previously Presented), please refer to above claim **4** accordingly.

19. Claims 6, 20, and 27 are rejected under 35 U.S.C. 103(a) as being unpatentable over Lehman in view of Gauthier, Liu, Xu, and David Lake (US 2004/0045003 A1) (hereinafter 'Lake')

20. **As to claim 6** (Previously Presented), Lehman discloses the method and the apparatus including means for further specifying *r* preemptive-tasks (e.g., Col. 9, lines 52-56), *r* being less than or equal to *n*, said task scheduler including a timer mechanism

that counts a specified period of time at which time if a preemptive-task is currently executing (e.g., Col. 35, lines 7-14) and continuing the execution of preemptive-task (e.g., Col. 9, line 64 through Col. 10, line 2)

But Lehman, Gauthier, Liu, and Xu do not specifically disclose the task's state is stored and execution is given to the task scheduler to schedule another task until a later time when the task scheduler restores the state of said preemptive-task.

However, in an analogous art, Lake discloses the task's state is stored and execution is given to the task scheduler to schedule another task until a later time when the task scheduler restores the state of said preemptive-task. However, in an analogous art, Lake discloses the task's state is stored and execution is given to said task scheduler to schedule another task until a later time when the task scheduler restores the state of said preemptive-task (e.g., Fig. 1; [0031]; [0026], lines 1-9; [0036], lines 1-6)

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to combine the teachings of Lehman, Gauthier, and Liu and Xu with the teachings of Lake in order to save/restore task control data during a preemptive-task interruption and task resumed in Lehman-Gauthier-Liu-Xu system.

The motivation is to have its stack pointer set to a pre-calculated worst-case value guaranteed to leave sufficient space in the stack beneath the stack pointer for any preemptive tasks for task suspended/restored operations as once suggested by Lake (i.e., Abstract)

21. **As to claim 20** (Previously Presented), please refer to above claim **6** accordingly.

22. **As to claim 27** (Previously Presented), please refer to above claim **6** accordingly.

Conclusion

23. Any inquiry concerning this communication or earlier communications from the examiner should be directed to Ben C. Wang whose telephone number is 571-270-1240. The examiner can normally be reached on Monday - Friday, 8:00 a.m. - 5:00 p.m., EST.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Tuan Q. Dam can be reached on 571-272-3695. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free). If you would like assistance from a USPTO Customer Service Representative or access to the automated information system, call 800-786-9199 (IN USA OR CANADA) or 571-272-1000.

/Ben C Wang/
Examiner, Art Unit 2192
September 10, 2008

/Tuan Q. Dam/
Supervisory Patent Examiner, Art Unit 2192